Algorithms and Data Structures 2
CS 1501

Spring 2022

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming deadlines:
  - Homework 6 due on 2/28
  - Lab 6 due on 3/4
  - Homework 7 due on 3/14
  - Assignment 1 due on 3/14
- Midterm exam on Wednesday 3/2
  - In-person, paper, closed book exam

Previous lecture …

Can we do better than Huffman's for lossless compression?

- Run-length encoding
- Shannon's entropy
- LZW

CourseMIRROR Reflections (most interesting)

- I found the discussion about the efficiency runtime interesting
- I enjoyed the brief intro to LZW compression as another method
- LZW is interesting to see how it builds new codewords
- The topic of entropy was most interesting.
- How buffers work
- The different ways to make the tries in Huffman compression
- The rate at which we can compress bits
- Information entropy and Kolmogorov complexity

CourseMIRROR Reflections (most confusing)

- The specific details with respect to the compression algorithms were most confusing today.
- I would like to go over entropy again and how it is calculated
- I was a bit confused on how Huffman encoding is the optimal method but we were still trying to find a method that was better
- How do we determine how many bits the codebook for LZW should have?
- The 12-bit code word and how it is generated.
- “Entropy as a measure of surprise”

Problem of the Day: Lossless Compression

- Input: A sequence of characters
  - *n* characters
  - each encoded as an 8-bit Extended ASCII
- Output: A bit string
  - of length less than 8*n
  - the original sequence can be fully restored from the bitstring

Problem of the Day

- Single-pass fixed-codeword-size Compression
- Input:
  - A sequence of *n* characters encoded using 8-bit Extended ASCII
  - A fixed codeword size, *L*
- Output: For each of the 2^L *possible codeword values,* determine a subsequence of the input that to be replaced by the codeword value
  - You can go over the input only once
  - the original sequence can be fully restored from the bitstring

LZW compression

- Initialize codebook to all single characters
  - e.g., character maps to its ASCII value
- While !EOF:
  - Match longest prefix in codebook
  - Output codeword
  - Take this longest prefix, add the next character in the file, and add the result to the dictionary with a new codeword

LZW compression example

- Compress, using 12 bit codewords:
  - TOBEORNOTTOBEORTOBEORNOT

Cur	Output	Add
T	84	TO:256
O	79	OB:257
B	66	BE:258
E	69	EO:259
O	79	OR:260
R	82	RN:261
N	78	NO:262
O	79	OT:263
T	84	TT:264
TO	256	TOB:265
BE	258	BEO:266
OR	260	ORT:267
TOB	265	TOBE:268
EO	259	EOR:269
RN	261	RNO:270
OT	263	--

LZW expansion

- Initialize codebook to all single characters
  - e.g., ASCII value maps to its character
- While !EOF:
  - Read next codeword from file
  - Lookup corresponding pattern in the codebook
  - Output that pattern
  - Add the previous pattern + the first character of the current pattern to the codebook

Note this means no codebook addition after first pattern output!

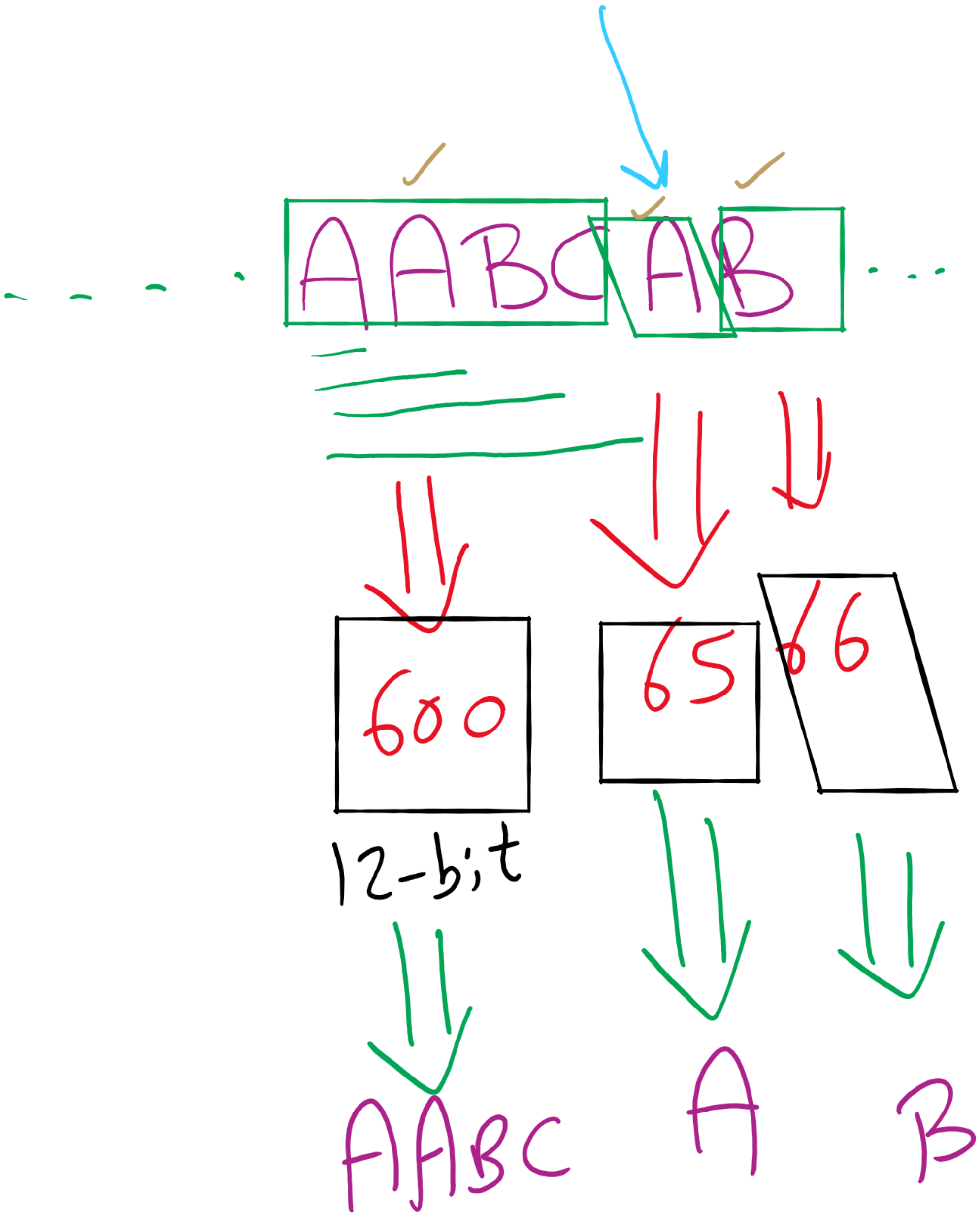
LZW expansion example

Cur	Output	Add
84	T	--
79	O	256:TO
66	B	257:OB
69	E	258:BE
79	O	259:EO
82	R	260:OR
78	N	261:RN
79	O	262:NO
84	T	263:OT
256	TO	264:TT
258	BE	265:TOB
260	OR	266:BEO
265	TOB	267:ORT
259	EO	268:TOBE
261	RN	269:EOR
263	OT	270:RNO

How does this work out?

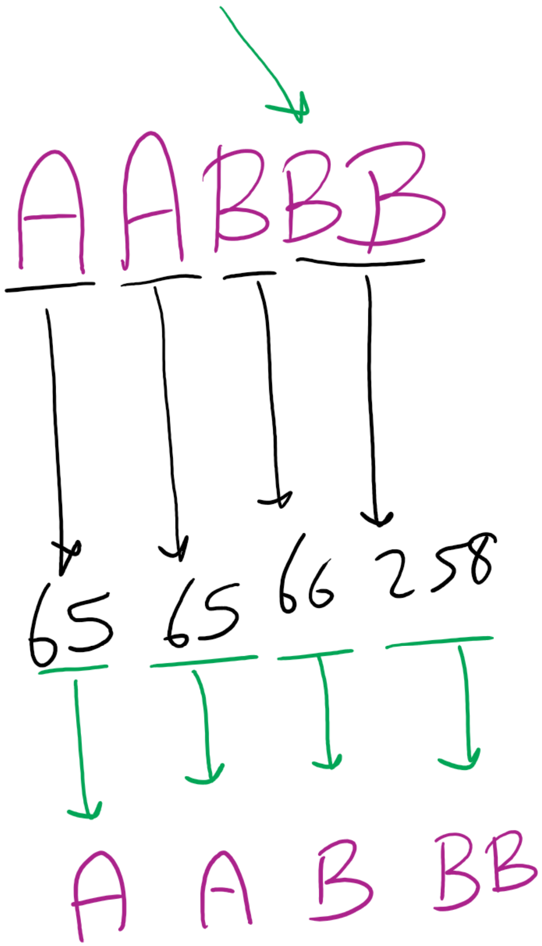
- Both compression and expansion construct the same codebook!
  - Compression stores character string → codeword
  - Expansion stores codeword → character string
  - They contain the same pairs in the same order
    - Hence, the codebook doesn’t need to be stored with the compressed file, saving space

LZW Example



A	65
B	66
AA	300
AAB	400
AABB	500
AABC	600
AABCA	601
AB	602

LZW Corner Case



A	65
AA	256
AB	257
BB	258

65	A
66	B
256	AA
257	AB
258	BB

|Codeword|

2

Prev output + 1st character of Prev output

Just one tiny little issue to sort out...

- Expansion can sometimes be a step ahead of compression...
  - If, during compression, the (pattern, codeword) that was just added to the dictionary is immediately used in the next step, the decompression algorithm will not yet know the codeword.
  - This is easily detected and dealt with, however

LZW corner case example

- Compress, using 12 bit codewords: AAAAAA

Cur	Output	Add
A	65	AA:256
AA	256	AAA:257
AAA	257	--

- Expansion:

Cur	Output	Add
65	A	--
256	AA	256:AA
257	AAA	257:AAA

LZW implementation concerns: codebook

- How to represent/store during:
  - Compression
  - Expansion
- Considerations:
  - What operations are needed?
  - How many of these operations are going to be performed?
- Discuss

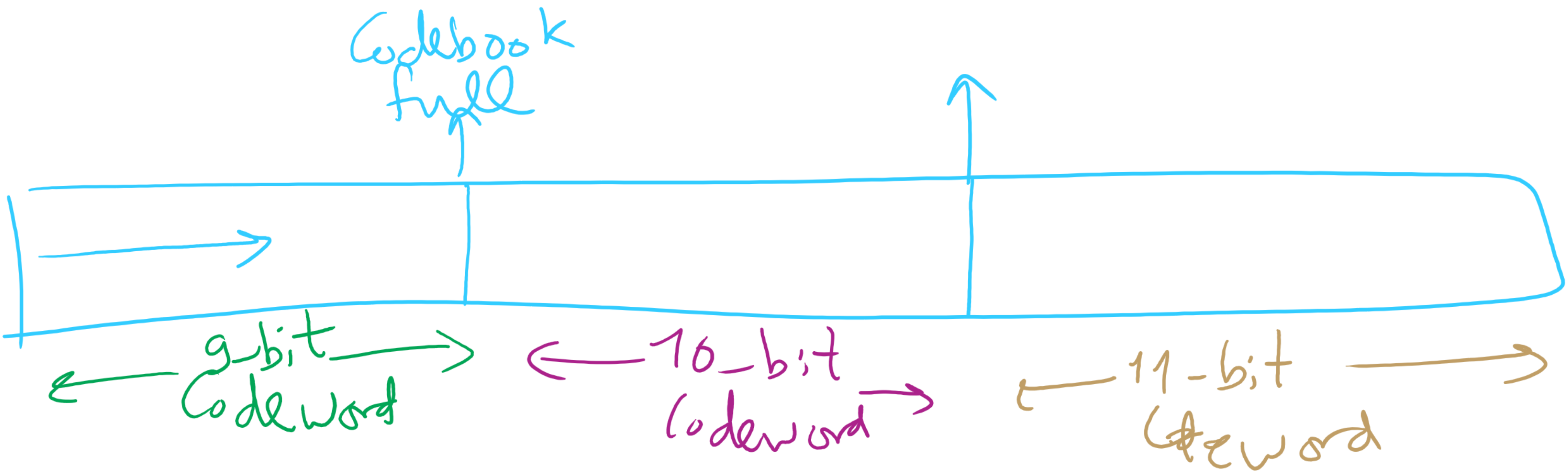
Further implementation issues: codeword size

- How long should codewords be?
  - Use fewer bits:
    - Gives better compression earlier on
    - But, leaves fewer codewords available, which will hamper compression later on
  - Use more bits:
    - Delays actual compression until longer patterns are found due to large codeword size
    - More codewords available means that greater compression gains can be made later on in the process

Variable width codewords

- This sounds eerily like variable length codewords…
    - Exactly what we set out to avoid!
- Here, we're talking about a different technique
- Example:
    - Start out using 9 bit codewords
    - When codeword 512 is inserted into the codebook, switch to outputting/grabbing 10 bit codewords
    - When codeword 1024 is inserted into the codebook, switch to outputting/grabbing 11 bit codewords…
    - Etc.

Adaptive Codeword Size



Even further implementation issues: codebook size

- What happens when we run out of codewords?
  - Only 2^n possible codewords for n bit codes
  - Even using variable width codewords, they can't grow arbitrarily large…
- Two primary options:
  - Stop adding new keywords, use the codebook as it stands
    - Maintains long already established patterns
    - But if the file changes, it will not be compressed as effectively
  - Throw out the codebook and start over from single characters
    - Allows new patterns to be compressed
    - Until new patterns are built up, though, compression will be minimal

The showdown you've all been waiting for...

HUFFMAN vs LZW

- In general, LZW will give better compression
    - Also better for compression archived directories of files
        - Why?
            - Very long patterns can be built up, leading to better compression
            - Different files don't "hurt" each other as they did in Huffman
                - Remember our thoughts on using static tries?

So lossless compression apps use LZW?

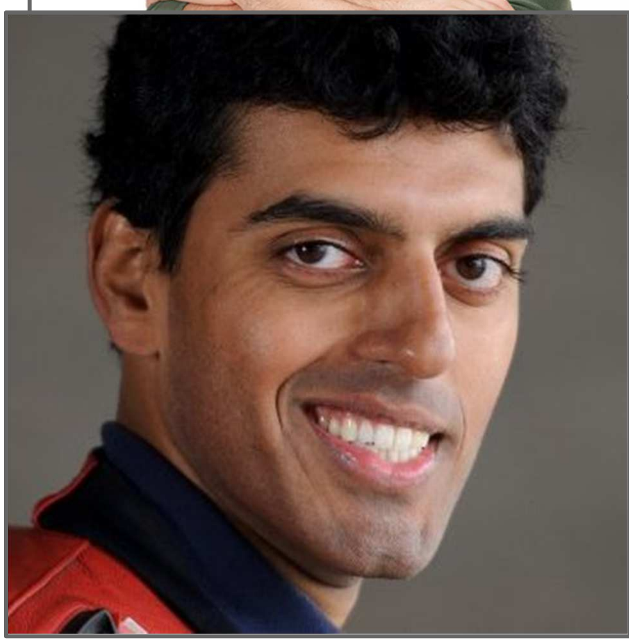
- Well, gifs can use it
  - And pdfs
- Most dedicated compression applications use other algorithms:
  - DEFLATE (combination of LZ77 and Huffman)
    - Used by PKZIP and gzip
  - Burrows-Wheeler transforms
    - Used by bzip2
  - LZMA
    - Used by 7-zip
  - brotli
    - Introduced by Google in Sept. 2015
    - Based around a " … combination of a modern variant of the LZ77 algorithm, Huffman coding[,] and 2nd order context modeling … "

DEFLATE et al achieve even better general compression?

- How much can they compress a file?
- Better question:
  - How much can a file be compressed by any algorithm?
- No algorithm can compress every bitstream
  - Assume we have such an algorithm
  - We can use to compress its own output!
  - And we could keep compressing its output until our compressed file is 0 bits!
    - Clearly this can't work
- Proofs in Proposition S of Section 5.5 of the text

A final note on compression evaluation

- "Weissman scores" are a made-up metric for Silicon Valley (TV)



Please submit your reflections by using the CourseMIRROR App

If you are having a problem with CourseMIRROR, please send an email to coursemirror.development@gmail.com